Parallel Processing of Multi-join Expansion_aggregate Data Cube Query in High Performance Database Systems

David Taniar
School of Business Systems
Monash University, Clayton Campus
Victoria 3800, AUSTRALIA
Email: David.Taniar@infotech.monash.edu.au

Rebecca Boon-Noi Tan
Gippsland School of Computing and IT
Monash University, Gippsland Campus
Victoria 3842, AUSTRALIA
Email: Rebecca.Tan@infotech.monash.edu.au

Abstract

Data cube queries containing aggregate functions often combine multiple tables through join operations. We can extend this to "Multi-Join Expansion_Aggregate" data cube queries by using more than one aggregate functions in "SELECT" statement in conjunction with relational operators. In parallel processing for such queries, it must be decided which attribute to use as a partitioning attribute, in particular, join attribute or cube-by. Based on the partitioning attribute, we introduce three parallel multi-join expansion_aggregate data cube query methods, namely Multi-join Partition Method (MPM), Expansion Partition Method (EPM) and Early Expansion Partition with Replication Method (EPRM). All three methods use the join attribute and cube-by as the partitioning attribute. Performance evaluation of the three parallel processing methods is also carried out and presented here.

1. Introduction

In recent years, heterogeneous decision support systems such as On-line Analytical Processing (OLAP) and data mining for analysing data in data warehouse have become topical issues in research community [1, 4, 8]. Queries involving aggregates are very common in database processing, especially in OLAP, and Data Warehouse [6]. These queries are often used as a tool for strategic decision making. With the vast amount of data growing rapidly in the data repository, efficient queries are critical and are set as high priority. We use parallelism techniques to achieve performance improvement of aggregate data cube queries. We are particularly interested in formulating efficient parallel processing methods for multi-join expansion_aggregate data cube queries especially in powerful PC processor environment. In this paper, we presented three parallel processing methods for multi-join expansion_aggregate data cube queries, *Multi-join Partition Method* (*MPM*), *Expansion Partition Method* (*EPM*) and *Early Expansion Partition with Replication Method* (*EPRM*).

2. Multi-Join Expansion_Aggregate Data Cube Query: A Background

Data cube aggregate queries normally involved a number of groups based on designated attributes where aggregate functions are carried out in each of the groups. Star schemas based on relational databases are often applied to data warehousing. ROLAP (Relational OLAP) that adapts the data architecture of the relational database and employs a star schema can execute high-speed retrievals and aggregations [5]. This star schema usually consists of a single fact table and a dimension table for each dimension as shown in Figure 1. The dimension tables Product, Location, Time, and Customer are connected with the fact table, Sales, by joining the keys Product_key (P#), Location_key (L#), Time_key (T#), and Customer_key (C#), respectively. Although this kind of star schema in a practical data warehouse application would typically have more than 6 tables or even hundreds. Typically, the fact table is much larger than any other table, such as the dimension table.

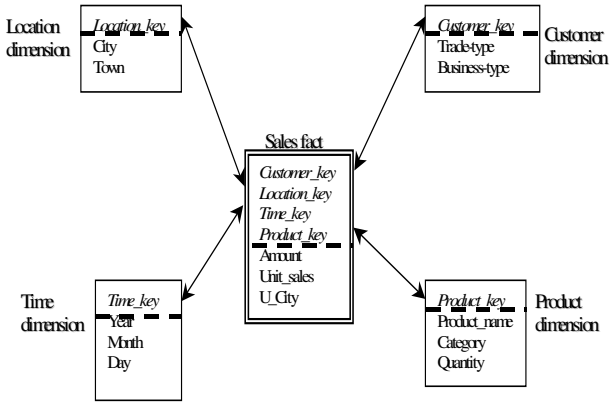

Figure 1. A simple star schema

Data cube queries containing aggregate functions often consist of multiple tables through join operations. Multi-join is using more than one SELECT list or joining more than one condition. Expansion_aggregate uses more than one aggregate function in the SELECT statement and also

Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02)
1087-4089/02 $17.00 © 2002 **IEEE**

uses the relational operator (that is, =, <, >, =<, >=, <>). We call this kind of query multi-join expansion_aggregate data cube queries.

For simplicity, we use the following query to give an illustration of multi-join expansion_aggregate data cube queries. This type of query involves more than one join and aggregate functions. Query is to "retrieve product number by their city location where the sales amount is less than or equal to 10,000 and the count of the related cities less than or equal to 3".

SELECT S.P#, S.U_City,
(**SELECT** (COUNT (S.U_City) <= 3)
FROM Sales S),
(**SELECT** (SUM (S.amount) <= 10,000)
FROM SALES S)
FROM Sales S, Product P, Location L
WHERE P.P# = S.P# and L.L# = S.L#
CUBE-BY S.P#, S.U_City

In the above Query, all Sales records are sum up based on the *P#* and *L#* attributes. After summing up this, the result is joined with tables Product and Location. In this paper, we focus on the cases where cube-by operation is performed before the join operation. Therefore, we will use the above Query as a theme example throughout this paper.

3. Parallel Multi-join Expansion_aggregate Query Processing methods

In this paper, we introduce three parallel processing methods for multi-join expansion_aggregate data cube queries, namely Multi-join Partition Method (MJM), Expansion Partition Method (EPM), Early Expansion Partition with Replication Method (EPRM). They are discussed in more detail in the following sections.

3.1 Multi-join Partition Method (MJM)

The Multi-join Partition method is influenced by the practice of parallel join algorithms, where raw records are first partitioned/distributed and allocated to each processor, and then each processor performs its operation [6]. This method is motivated by fast message passing multi processor systems.

The Multi-join Partition method comprised of two phases: *distribution* phase and *cube-by with multi-join* phase. Using Query, the three tables to be joined are Sales, Product and Location based on attributes *P#* and *L#*, and the cube-by will be based on table Sales. For simplicity of notation, the table which becomes the basis for cube-by is called fact table *F* (e.g. table Sales), and the other tables are called D_1 *and* D_2 (e.g. tables Product and Location). From now on, we will refer them as tables *F,* D_1 *and* D_2.

In the distribution phase, raw records from three tables (i.e. tables *F,* D_1 *and* D_2) are distributed based on the join/cube-by attributes according to two data partitioning functions. In the first partitioning function, we allocate product numbers of a certain range to each processor. For example, product numbers (attribute *P#*) *p1-p100* to processor 1, product numbers *p101-p200* to processor 2, product numbers *p201-p300* to processor 3, and so on. We need to emphasize that the raw records within the three tables *F,* D_1 *and* D_2 are all distributed. As a result, for example, processor 1 will have records from the Sales table with *P#* between *p1* and *p100*, inclusive, as well as records from the Product table with *P# p1-p100*. The process is repeated for the second partitioning function for records of *p1-p100,* whereby *p1-p100* is distributed according to location numbers (attribute *L#*) *L1-L25*, *L26-L50*, *L51-L75*, and so on among the processors. This is then applied to *p101-p200*, and so on. This distribution scheme is commonly used in parallel join, where raw records are partitioned into buckets based on an adopted partitioning scheme like the above range partitioning scheme [6].

Once the distribution is completed, each processor will have records within specific group range identified by the cube-by/join attribute. Subsequently, the second phase (the cube-by with multi-join phase) calculates the aggregate values on each group. Aggregating in each processor can be carried out through a sort or a hash function. After table *F* is grouped in each processor, it is joined with tables D_1 *and* D_2 in the same processor. After joining, each processor will have a local query result. The final query result is a union of all sub-results produced by each processor.

Figure 2 shows an illustration of the Multi-join Partition method. Notice that partitioning is done to the raw records of three tables *F,* D_1 *and* D_2, and aggregate operation of table *F,* and then join with tables D_1 *and* D_2 in each processor is carried out after the distribution phase.

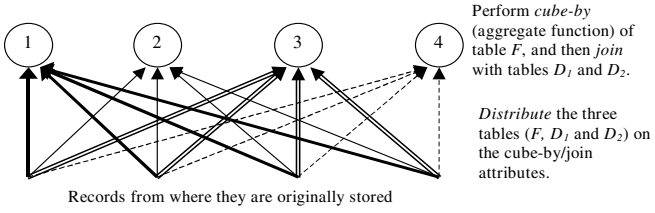

Figure 2. Multi-join Partition method (MJM)

3.2 Expansion Partition Method (EPM)

Expansion Partition Method performs the cube-by operation first before anything else (e.g. distribution). The Expansion Partition Method comprised of three phases: (*i*)

local clustering phase, (*ii*) *distribution* phase, and (*iii*) *final-amass* phase.

In the local clustering phase, each processor performs its cube-by operation and calculates its local aggregate values on records of table *F*. In this phase each processor groups local records *F* according to the designated cube-by attribute and performs the aggregate function. Using the same example as that in the previous section, one processor may produce, for example, (*p1*, 2000) and (*p140*, 7000), and another processor (*p101*, 8000) and (*p140*, 4000). The numerical figures indicate the SUM(amount) of each Sales.

In the second phase (i.e. distribution phase), the results of local aggregates from each processor, together with records of tables D_1 *and* D_2, are distributed to all processors according to two partitioning function. The two partitioning function are based on the join/cube-by attributes, which in this case is attribute *P#* of tables Product and Sales and also attribute *L#* of tables Location and Sales. Again using the same partitioning function in the previous section, *P#* of *p1-P100* are to be distributed to processor 1, *P#* of *p101-p200* to processor 2, and so on.

In the third phase (i.e. final-amass phase), two operations are carried out - final aggregate or grouping of *F*, and join with D_1 *and* D_2. The final clustering can be carried out by merging all temporary results obtained in each processor. The way it works can be explained as follows. After local aggregates are formulated in each processor, each processor then distributes each of the groups to another processor depending on the adopted distribution function. Once the distribution of local results based on a particular distribution function is completed, global aggregation in each processor is simply done by merging all identical product number (*P#*) into one aggregate value. For example, processor 2 will merge (*p140*, 7000) from one processor and (*p140*, 4000) from another to produce (*p140*, 11000) which is the final aggregate value for this product number.

After global aggregation results are obtained, it then joins tables D_1 *and* D_2 in each processor. Figure 3 shows an illustration of this scheme. There are several note worthy points that are of interest. First, records *F* in each processor are aggregated/grouped before distributing them. Consequently, communication costs associated with table *F* can be expected to reduce depending on the cube-by selectivity factor. This method is expected to improve the Multi-join Partition Method. Second, we observe that if the number of groups is less than the number of available processors, not all processors can be exploited; reducing the capability of parallelism. And lastly, records from tables D_1 *and* D_2 in each processor are all distributed during the second phase. In other words, there is no filtering mechanism applied to D_1 *and* D_2 prior to distribution. This can be inefficient particularly if D_1 *and* D_2 are very large. To avoid the problem of distributing D_1 *and* D_2, we will introduce another method in the next section.

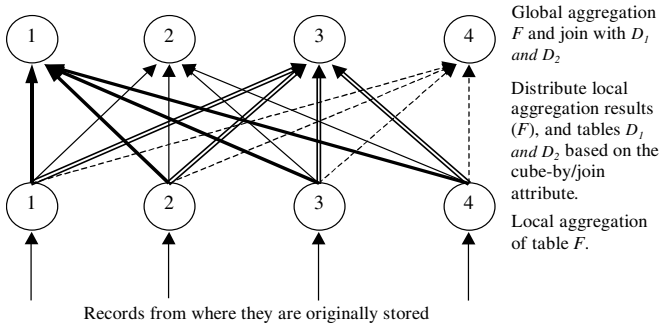

Figure 3. Expansion Partition Method (*EPM*)

3.3 Early Expansion Partition with Replication Method (*EPRM*)

Early Expansion Partition with Replication Method is similar to the Expansion Partition method. The similarity is due to the cube-by processing to be done before the distribution phase. However, the difference is pointed by the keyword "*Replication*" in this method, as opposed to "*Partition*". The Early Expansion Partition with Replication Method, which also comprised of three phases, works as follows. The first phase that is the local clustering phase, is exactly the same as that of the Expansion Partition method where local aggregate is performed to table *F* for each processor. The main difference is in phase two. Using the "Replication" method, the local aggregate results obtained from each processor are replicated to all processors. Tables D_1 *and* D_2 are not moved at all from where they are originally stored.

The third phase, the final-amass phase, is basically similar to that of the " Partition" method, where the local aggregates from all processors are merged to obtain global aggregate and then joined with D_1 *and* D_2. On closer inspection, we can find a difference between the two Expansion methods. In the "Replication" method, after the replication phase, each processor will have local aggregate results from all processors. Consequently, processing global aggregates in each processor will produce the same results, and this can be inefficient as no parallelism is employed. However, joining and global aggregation processes can be done at the same time. First, hash local aggregate results from *F* to obtain global aggregate values, and then hash and probe the fragment of tables D_1 *and* D_2 to produce final query result. The minor snag is that many of the global aggregate results will have no match with local tables D_1 *and* D_2 in each processor.

Figure 4 gives a graphical illustration of the scheme. It looks very similar to Figure 3, except that in the replication

phase, the arrows shown are thicker to emphasize the fact that local aggregate results from each processor are replicated to all processors, not distributed.

Apart from the facts that the non cube-by tables (tables D_1 *and* D_2) are not distributed and the local aggregate results of table F are replicated, and assuming that tables D_1 *and* D_2 are uniformly distributed to all processors initially (that is round-robin data placement is adopted in storing records D_1 *and* D_2), there will be no skew problem in the joining phase.

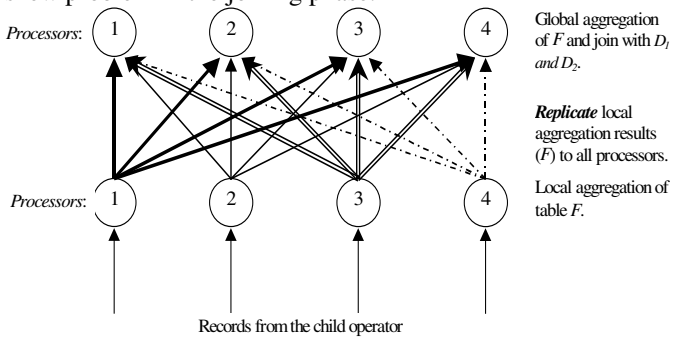

Figure 4. Early Expansion Partition with Replication (EPRM)

4. Performance Evaluation

In this paper, we want to study the behaviour of the methods described above and to compare their performances. In order to do that, a sensitivity analysis through a simulation technique is carried out. This sensitivity analysis is performed through varying the performance parameters. The parameters of the study basically comprised of parameters known by the system as well as the data, such as parameters related to the query, unit time costs, and communication costs as shown in Table 1.

4.1 Result of Experiment

It is of interest to find out which method performs best. The result is based on the above parameters. Table 2 shows the three types of methods with their phases that we have mentioned before.

The graphs in Figure 5 show a comparative performance between the three parallel methods by varying the Cube-By selectivity ratio (i.e. number of groups produced by the query). The selectivity ratio varies from 0.0000001 to 0.01. With 100 million records as input, the selectivity ratio of 0.0000001 produces 10 groups, whereas the other end of selectivity ratio of 0.01 produces 1 million groups. The machine in the experiment consists of 64 processors. The graphs also show the results when variation on parameters was applied.

Description	Value
System and Data Parameters	
Number of processors	64 *processors*
MIPS of the processor	450 *Mips*
Fact Table size (Sales)	10 *GB*
Number of records in table *F*	100 *Million records*
Dimensional Table size (Product)	30 *MB*
Number of records in table D_1	30 *thousand records*
Dimensional Table size (Location)	2.5 *KB*
Number of records in table D_2	25 *records*
Page size	4 *KB*
Maximum hash table entries	10,000 *entries*
Query Parameters	
Projectivity ratio of the aggregation	0.15
Selectivity ratio of local aggregate in a processor	0.0000001 *to* 0.01
Selectivity ratio of local aggregate in a node	0.0000001 *to* 0.01
Selectivity ratio of global aggregate	0.0000001 *to* 0.01
Join selectivity ratio	0.00000025
Time Unit Costs	
Effective time to read a page from disk	3.5 *ms*
Time to read a record	300/*Mips*
Time to write a record	100/*Mips*
Time to compute hash value	400/*Mips*
Time to add a record to current aggregate value	300/*Mips*
Time to compute destination	10/*Mips*
Time to compare a record with a hash table entry	100/*Mips*
Communication Unit Costs	
Message protocol cost per page	1000/*Mips*
Message latency for one page	1.3 *ms*

Table 1. Parameters

	Multi-join Partition Method	Expansion Partition Method	Early-expansion Partition Method
Phase One	Distribution	Local clustering	Local clustering
Phase Two	Cube-by with multi-join	Distribution	Replication
Phase Three		Final-amass	Final-amass

Table 2. Which method is able to give the best performance?

Using the Multi-join Partition method, more extensive data processing occurs during the first phase. In the first phase, the raw records are scanned and then distributed equally to each processor based on certain arrangement. Therefore the major cost of the method is on the scanning, loading and transferring data to every processor, and after that each processor is loaded with equal task and grouped data. Therefore, in the second phase, the total cost is minor unless the maximum capacity of each processor is exceeded. As the consequence, although the process of data transfer, aggregation and join and other processes occur after that, these have minimal effect on the total cost. This conforms to the graph which shows the total performance

is hardly affected. Factors like faster processors (4 times), faster communications network (4 times) and bigger memory (10 times) have more influences on the second phase of the method (see Figure 6).

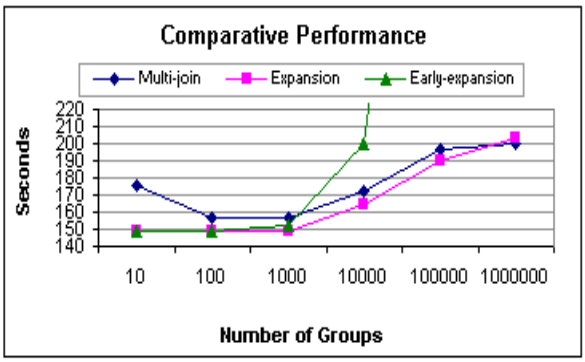

Figure 5. Comparative Performance

In the Expansion Partition Method, data scanning and loading are still the major cost factors. Besides that, the second major data cost is data transfer and reading/writing of overflow of bucket process. This is due to the process of early clustering occurrence. In general, applying faster processors, network and bigger memory in this method does not have much impact on the overall performance. This is almost similar to the Multi-join Partition Method. However the performance remained steady as the number of groups increases to 1000, and then the performance starts to decline. This conforms to the logic, that when number of groups produced increases, the data volume of data of the second and third phase also increases (also see Figure 6).

The result obtained using the Early Expansion Partition with Replication Method is different compared to the two previous methods. In this method, the major costs exist in all three phases of the method. The major costs are the data scanning and loading, data transfer and aggregate and join processes. During the first phase, the data scanning and loading is the major cost. This is so as the result of the early clustering process. Then during the second phase, the data transfer contributes to the decrease of performance, as all data are sent to all processor for replication. Finally during the third phase, pooling all data to all processors also contributes to the decrease of the performance, especially when number of groups continues to grow.

4.2 Discussions

Expansion Partition method shows the best performance if the number of groups produced tends to be large, especially above 1000 groups. It is quite interesting to note that in both the Multi-join Partition and Expansion Partition methods, the time (in seconds) taken to process the result remain fairly constant within the range of 10 to 1000 number of groups. However, this applies only within the graphs in Figure 6 for faster CPU, faster network and faster disk but not for the graph for bigger main-memory.

As the variation is being applied, for example the variation for faster disk, the overall performance of all the methods improves significantly. This applies where the number of groups ranges from 0 to 1000. However, when the number of groups is greater than 1000, there is a significant decline of performance in the Early Expansion Partition with Replication method as compared to the other two methods. This also applies for factors such as faster CPU, faster network and bigger memory.

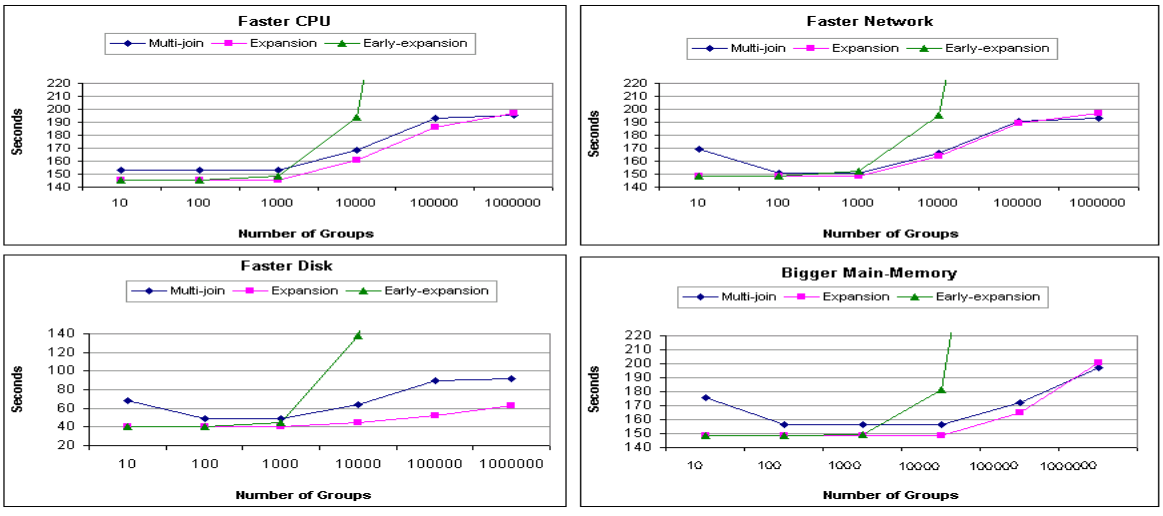

Figure 6. Varying By Selectivity Ratio

5. Related Work

There is a vast pool of literature applying parallel processing techniques to relational data base systems (e.g. [6]). From this literature has emerged the notion that highly-parallel, shared-nothing architectures can yield much better performance than equivalent closely-coupled systems [6]. Various methods have been developed over the years to distribute data across sites. With the popularity of decision support systems such as data warehouse, OLAP etc, there is a critical demand of efficient query processing and for quick response time. Datta, Moon & Thomas [5] offered a shared-nothing approach to parallelise an OLAP database system and its focus is primarily on data structures enabling proper fragmentation and allocation of data.

Others work on materialized views provide the need for an efficient, mostly read-only access to aggregates in a multidimensional context [1, 7]. Lo, Hua & Young [9] presented a multidimensional data partitioning technique using multiple attributes for partitioning. Shukla, Deshpande & Naughton [14] also presented the aggregate selection for multi-cube data models, which compute aggregates over multiple cubes.

There has been extensive work on different multi-join methods (e.g. [10]). Recent work are attracted to process certain kinds of star queries [13, 15], top N queries [3], TID hash joins [11], special index structures such as bitmap indices [12] or functional joins [2]. In this paper, we are particularly interested in formulating efficient parallel processing methods for multi-join expansion_aggregate data cube queries.

6. Conclusion and Future work

In this paper, we have presented three parallel processing methods for multi-join expansion_aggregate data cube queries, Multi-join Partition Method (MPM), Expansion Partition Method (EPM) and Early Expansion Partition with Replication Method (EPRM). These methods differ in ways in which the query tables are distributed.

From our study it is concluded that the Expansion Partition Method is preferred to the two methods. Our performance evaluation results show that the variation in faster disk has the potential to produce the most efficient performance in all the situations investigated. In addition, increasing the number of processors, increasing the speeding of the CPU and adding bigger memory are some of the other techniques suggested that can be applied as the number of groups grow.

Our future work in this field includes further refinement of the methods, implementation and testing of the methods for OLAP operations. As this type of applications normally involves vast amount of data, parallelism technique is important and necessary in order to keep the performance level acceptable.

References

[1] Albrecht et al, "Management of multidimensional Aggregates for efficient Online Analytical Processing", *Proceedings of International Database Engineering and Applications Symposium (IDEAS)*, Montreal, 1999.

[2] Braumandl R., Claussen J. and Kemper A., "Evaluating functional joins along nested references sets in object-relational and object-oriented databases", *Proceedings of the International Conference on Very Large Data Bases (VLDB),* pp. 110-121, New York, USA, August 1998.

[3] Carey M. and Kossmann D., "Reducing the braking distance of an SQL query engine", *Proc. of the Intl Conf. on Very Large Data Bases (VLDB),* pp. 158-169, 1998.

[4] Chan C.Y. and Ioannidis Y.E., "Hierarchical Cubes for Range_Sum Queries", *Proc. of Intl. Conf. on Very Large Data Bases (VLDB)*, Edinburgh, pp. 675-686, 1999.

[5] Datta A. and Moon B., "A case for parallelism in data warehousing and OLAP", *Proceedings of 9th International Workshop on Database Systems Applications*, 1998

[6] DeWitt, D.J. and Gray, J., "Parallel Database Ssytems: The Future of High Performance Database Systems", *Communication of the ACM*, 35(6), pp. 85-98, 1992.

[7] Harinarayan V., Rajaraman A. and Ullman. J., "Implementing Data Cubes Efficiently", *Proceedings of ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, pp. 205-216, June 1996.

[8] Lee S.Y, Ling T.W. and Li H.G., "Hierarchical Compact Cube for Range_Max Queries", *Proc. of Intl. Conf. on Very Large Data Bases (VLDB)*, Cario, pp. 232-241, 2000.

[9] Lo Y., Hua K., Young H., "A General Multidimensional Data Allocation Method for Multicomputer Database Systems", *Proc. of 8th Intl. Conf. on Database and Expert Systems Applications (DEXA)*, Toulouse, France, 1997.

[10] Lu, H., Shan M. C. and Tan K. L., "Optimization of Multi-Way join Queries for Parallel Execution", *In Proceedings of the International Conference on Very Large Data Bases (VLDB),* pp. 549-560, Barcelona, Spain, August 1990.

[11] Marek R.and Rahm E., "TID hash joins", *Proceedings of International Conference on Information and Knowledge Management (CIKM),* pp. 42-49, Maryland, USA, 1994

[12] O'Neil P. and Graefe G., "Multi-Table Joins Through Bit-mapped Join Indices", *Proceedings of ACM SIGMOD*, Record 24 (3), 1995.

[13] Shiefer et al, "IBM's DB2 Universal Database demonstration", *Proc. of the Intl. Conference on Very Large Data Base (VLDB),* New York, USA, pp. 703, 1998

[14] Shukla A., Deshpande P. and Naughton J., "Materialized View Selection for Multi-cube Data Models", *Proc. of the Intl. Conference on Extending Database Technology (EDBT*), LNCS, Springer-Verlag, pp. 269 – 284, 2000

[15] Stohr T., Martens H. and Rahm E., "Multi-Dimensional Database Allocation for Parallel Data Warehouses", *Proc. of the 26th International Conference on Very Large Data Base (VLDB),* Cario, Egypt, pp. 273-284, 2000